

LA-UR-18-25478

Approved for public release; distribution is unlimited.

Title: Malicious Docs: Malware Analysis Day 7

Author(s): Pearce, Lauren

Intended for: Presentation for 2 week malware analysis class

Issued: 2018-06-21

Disclaimer:

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the Los Alamos National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396. By approving this article, the publisher recognizes that the U.S. Government retains nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

Malicious Docs

Malware Analysis Day 7

Obfuscated Javascript

Why?

- Exploit kits (or other malware) looking for vulnerable browsers to provide a way in.
 - Web Drive By
 - Watering Hole
 - Phishing
- Javascript tagging along on other content
 - PDF documents
 - Flash files

Options for Analysis

- Static analysis, reviewing the obfuscated code by hand
- Dynamic analysis, letting the code run in a protected sandbox so it can deobfuscate itself

3 Steps

1. Clean-up the Javascript in question to make it easier to understand, and to enable better setting of breakpoints
2. Modify the Javascript to run in an interpreter (Firefox, Chrome, or command-line versions of the browser interpreter such as Spidermonkey or V8)
3. Run the code, modifying statements to include print statements or setting breakpoints

Formatting the JS

- Format it to look pretty. I use PDFStreamDumper since it's already in my VM, but there are other options. Here's how to use PDFStreamDumper:
 1. Copy the js into your clipboard
 2. Open PDFStream Dumper, select "JavaScriptUI" from the top toolbar
 3. Paste the js into the text editor and select all
 4. Click "Format_Javascript"
 5. Copy and paste your formatted javascript into a new file.

Eval

- Eval is a function that evaluates a string as if it is code, then returns the results. This is useful to us, because we often want to see the code that is being passed into eval.
- Two basic methods to see what's being passed to eval
 - Debugger – browser based or other.
 - Print statements, `console.log` in js
- Figuring out what's going to eval is rarely ALL that you have to do, but it's a good step anytime you see an eval call.

Console.Log

- To use this method, we have to first turn our js into html, then add in a print statement, and lastly run the code in an interpreter.
 1. Add ``<html>`` and ``<script>`` tags to the beginning of the code and corresponding end tags to the end.
 2. Replace the eval statement with "console.log"
 3. Save as a .html
 4. Open your file in Firefox
 5. In firefox, go to Menu --> Developer --> Web Console
 6. In the pane that opens, you will see either error code or deobfuscated code

Demo

00_jsexample

Malicious Office Docs

Macros

Office Documents

- Allow scripting via VBA and are capable of interacting with the windows OS
- Major vector of infection

Office File Formats

- Binary file format (doc, ppt, xls, etc.)
 - Data within the file is stored in one or more streams, which contain data and necessary metadata.
- OpenXML file format (docx, pptx, xlsx, etc.)
 - Follow the OpenXML standard. Can be parsed by anything capable of parsing XML
 - The XML is stored in compressed archives and must be decompressed. 7z works.
 - VBA macros typically stored in binary OLE file called vbaProject.bin

VBA

- Visual Basic for Applications – uses same runtime library and syntax as VB, but will only function within the hosting application.
- VBA can be used to inject shellcode, interact with Windows OS resources, and to interact with web resources to pull down malware.

Tools

- OfficeMalScanner
 - A Windows command-line tool for parsing and analyzing binary and xml Office formats. It can find, identify and extract shellcode, embedded OLE streams, PE files, or non-standard functionality within Office documents.
- oletools
 - A set of python tools for interacting with OLE streams
 - Included in the REMnux VM – explore what's available in /opt/remnux-oletools

OfficeMalScanner

- Usage: `OfficeMalScanner <file> <scan | info | inflate> <brute> <debug>`
- Options:
 - scan - scans for shellcode and encrypted PE-files
 - info - dumps OLE structures and saves VB-Macro code
 - inflate - decompresses MS 2007 docs into a temp dir
- Switches: (only enabled if "scan" option was selected)
 - brute - brute force search for encrypted stuff
 - debug - prints out disassembly resp hexoutput if a heuristic was found

OfficeMalScanner

```
C:\Reversing\OfficeMalScanner>officemalscanner XB714691IE.doc inflate
```

```
+-----+
| OfficeMalScanner v0.61 |
| Frank Boldewin / www.reconstructor.org |
+-----+
```

```
[*] INFLATE mode selected
[*] Opening file XB714691IE.doc
[*] Filesize is 47441 (0xb951) Bytes
[*] Microsoft Office Open XML Format document detected.
```

Found 15 files in this archive

```
[Content_Types].xml ----- 1503 Bytes ----- at Offset 0x00000000
_rels/.rels ----- 590 Bytes ----- at Offset 0x000003c8
word/_rels/document.xml.rels ----- 1071 Bytes ----- at Offset 0x000006e8
word/document.xml ----- 4094 Bytes ----- at Offset 0x00000956
word/vbaProject.bin ----- 14336 Bytes ----- at Offset 0x00000e08
word/media/image1.gif ----- 27313 Bytes ----- at Offset 0x0000253d
word/_rels/vbaProject.bin.rels ----- 277 Bytes ----- at Offset 0x00009021
word/theme/theme1.xml ----- 7043 Bytes ----- at Offset 0x0000911c
word/vbaData.xml ----- 1749 Bytes ----- at Offset 0x00009762
word/settings.xml ----- 7760 Bytes ----- at Offset 0x00009978
docProps/app.xml ----- 731 Bytes ----- at Offset 0x0000a18e
word/styles.xml ----- 28883 Bytes ----- at Offset 0x0000a449
docProps/core.xml ----- 636 Bytes ----- at Offset 0x0000afc8
word/fontTable.xml ----- 1255 Bytes ----- at Offset 0x0000b24b
word/webSettings.xml ----- 484 Bytes ----- at Offset 0x0000b43a
```

Content was decompressed to C:\Users\TRACER~1\AppData\Local\Temp\DecompressedMsOfficeDocument.

Found at least 1 ".bin" file in the MSOffice document container.
Try to scan it manually with SCAN+BRUTE and INFO mode.

```
C:\Reversing\OfficeMalScanner>officemalscanner vbaproject.bin info
```

```
+-----+
| OfficeMalScanner v0.61 |
| Frank Boldewin / www.reconstructor.org |
+-----+
```

```
[*] INFO mode selected
[*] Opening file vbaproject.bin
[*] Filesize is 14336 (0x3800) Bytes
[*] Ms Office OLE2 Compound Format document detected
```

[Scanning for VB-code in VBAPROJECT.BIN]

ThisDocument

VB-MACRO CODE WAS FOUND INSIDE THIS FILE!
The decompressed Macro code was stored here:

-----> C:\Reversing\OfficeMalScanner\VBAPROJECT.BIN-Macros

olevba

```
remnux@remnux:~/Desktop$ olevba.py XB714691IE.doc
olevba 0.27 - http://decalage.info/python/oletools
Flags      Filename
```

```
-----
OpX:MASI--- XB714691IE.doc
```

```
(Flags: OpX=OpenXML, XML=Word2003XML, MHT=MHTML, M=Macros,
strings, B=Base64 strings, D=Dridex strings, ?=Unknown)
```

```
=====
FILE: XB714691IE.doc
Type: OpenXML
```

```
-----
VBA MACRO ThisDocument.cls
in file: word/vbaProject.bin - OLE stream: u'VBA/ThisDocum
```

```
-----
Sub Auto_Open()
    h
End Sub
Sub h()
    Dim MY_FILEDIR, MY_FILEDIR, MY_FILDIR
    MY_FILEN = "ntusersc.ps1"
    MY_FILE = "ntusersss.bat"
    MY_FIL = "ntuserskk.vbs"
    MY_FILEDIR = ActiveDocument.Path + "\ntusersc.ps1"
```

| Type | Keyword | Description |
|------------|---------------------------------------|---|
| AutoExec | AutoOpen | Runs when the Word document is opened |
| AutoExec | Auto_Open | Runs when the Excel Workbook is opened |
| AutoExec | Workbook_Open | Runs when the Excel Workbook is opened |
| Suspicious | Open | May open a file |
| Suspicious | Chr | May attempt to obfuscate specific strings |
| Suspicious | CreateObject | May create an OLE object |
| Suspicious | Shell | May run an executable file or a system command |
| Suspicious | WScript.Shell | May run an executable file or a system command |
| Suspicious | Run | May run an executable file or a system command |
| Suspicious | PowerShell | May run PowerShell commands |
| Suspicious | New-Object | May download files from the Internet using PowerShell |
| Suspicious | System.Net.WebClient | May download files from the Internet using PowerShell |
| Suspicious | DownloadFile | May download files from the Internet using PowerShell |
| Suspicious | Output | May write to a file (if combined with Open) |
| Suspicious | Print # | May write to a file (if combined with Open) |
| IOC | http://162.243.234.167:8080/gr/4.exe' | URL |
| IOC | 162.243.234.167 | IPv4 address |
| IOC | 1.1.2.2 | IPv4 address |
| IOC | ntusersc.ps1 | Executable file name |
| IOC | ntusersss.bat | Executable file name |
| IOC | ntuserskk.vbs | Executable file name |
| IOC | 4.exe | Executable file name |

Demo

OfficeMalScanner and oleTools

Malicious PDFs

PDFs and Malware

- You do NOT need to have a complete understanding of the PDF format to understand how malware uses a PDF.
 - The PDF documentation is well over 700 pages...
- What is a PDF?
 - A binary (most common) or ascii file written in a specific format. That format can be rendered as a document by a PDF reader.
- How can malware use a PDF?
 - Some program (typically adobe reader) must interpret the data provided in the PDF file to present the user with a well formatted document to look at. Vulnerabilities in that program can be exploited by a well crafted PDF.

PDF Format

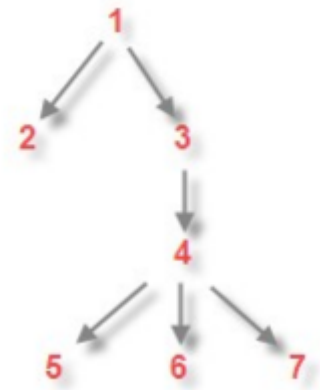
- Indirect Objects

- Have an index number, a version number, and content.
- Content is between the keywords `obj` and `endobj`
- Can refer to other indirect objects by using their index and reference number
 - `5 0 R` <- Reference to object 5 version 0

```
5 0 obj  
...  
endobj
```

- Logical Structure

- References between indirect objects form a tree – the logical structure of a PDF.
- The root element is identified by a `/Root` tag
- Physical structure is the order the objects occur in the document – completely different.



PDF Format

- Stream Objects
 - A type of indirect object that contains data between the keywords stream and endstream.
 - Often compressed – PDFs refer to compression algorithms as filters.
 - A stream can be compressed by more than one filter.

```
5 0 obj<</Subtype/Type1C/Length 5416/Filter/FlateDecode>>stream
H%|T}T#W#Y!d&"FI#Å%NFW#âC
...
endstream
endobj
```


PDF Format - Content

- Static:
 - Text blocks and styles
 - Character encodings and font selection
 - Multimedia support instructions
- Dynamic:
 - Embedded JavaScript
 - Dynamic action triggers (e.g., "On Open")
 - Interaction with network resources (e.g., retrieving information from a web resource by URL)

Dictionary Entries

- /<keyword> - The key to a dictionary that instructs the PDF reader to perform some action. These are often referred to as “tags”.
- /OpenAction means to perform the action in the specified stream.
 - Frequently used to execute javascript

```
1 0 obj
<<
  /Type /Catalog
  /Outlines 2 0 R
  /Pages 3 0 R
  /OpenAction 7 0 R
>>
endobj
```

Tags of Interest

- /OpenAction and /AA: specifies a script or action to take place automatically
- /Names, /AcroForm, /Action: can specify or launch scripts
- /JavaScript: JavaScript to run
- /Launch: launches a program or document
- /URI: access a resource by URL
- /SubmitForm and /GoToR: send data to a specified URI
- /RichMedia: embed Flash content in a PDF
- /ObjStm: an object stream which can be used to hide objects

PDF Stream Dumper

PDFStreamDumper - http://sandsprite.com FileSize: 45 Kb LoadTime: 0.156 seconds

Load Exploits_Scan Javascript_UI Unescape_Selection Manual_Escapes Update_Current_Stream Goto_Object Search_For Find/Replace Tools Help_Videos

21 Objects

- 1 HLen: 0x58
- 4 HLen: 0x16
- 2 HLen: 0x2D
- 3 HLen: 0x30
- 5 0x166-0x7EB
- 6 HLen: 0x5F
- 7 0x8B1-0xA5D
- 8 HLen: 0x8
- 9 HLen: 0x25
- 10 0xAE8-0xB2C
- 11 HLen: 0x7
- 12 HLen: 0x12
- 13 HLen: 0x6
- 14 HLen: 0x6
- 15 HLen: 0x33D
- 16 HLen: 0xF7
- 17 0x10C3-0xADF7
- 18 HLen: 0x9
- 23 HLen: 0xF
- 24 0xAE6D-0xAFEB
- 0 HLen: 0x224

```
<<  
  /Type /Catalog  
  /Version /1.4  
  /Pages 2 0 R  
  /OpenAction 3 0 R  
  /AcroForm 23 0 R  
>>
```

Text HexDump Stream Details


1 Decompression Errors
stream # 17 org sz = (0x58)

Errors Search Debug (2)

Shell PDF Path C:\Users\Tracer Fire\Desktop\,sample1.pdf ... Load Abort

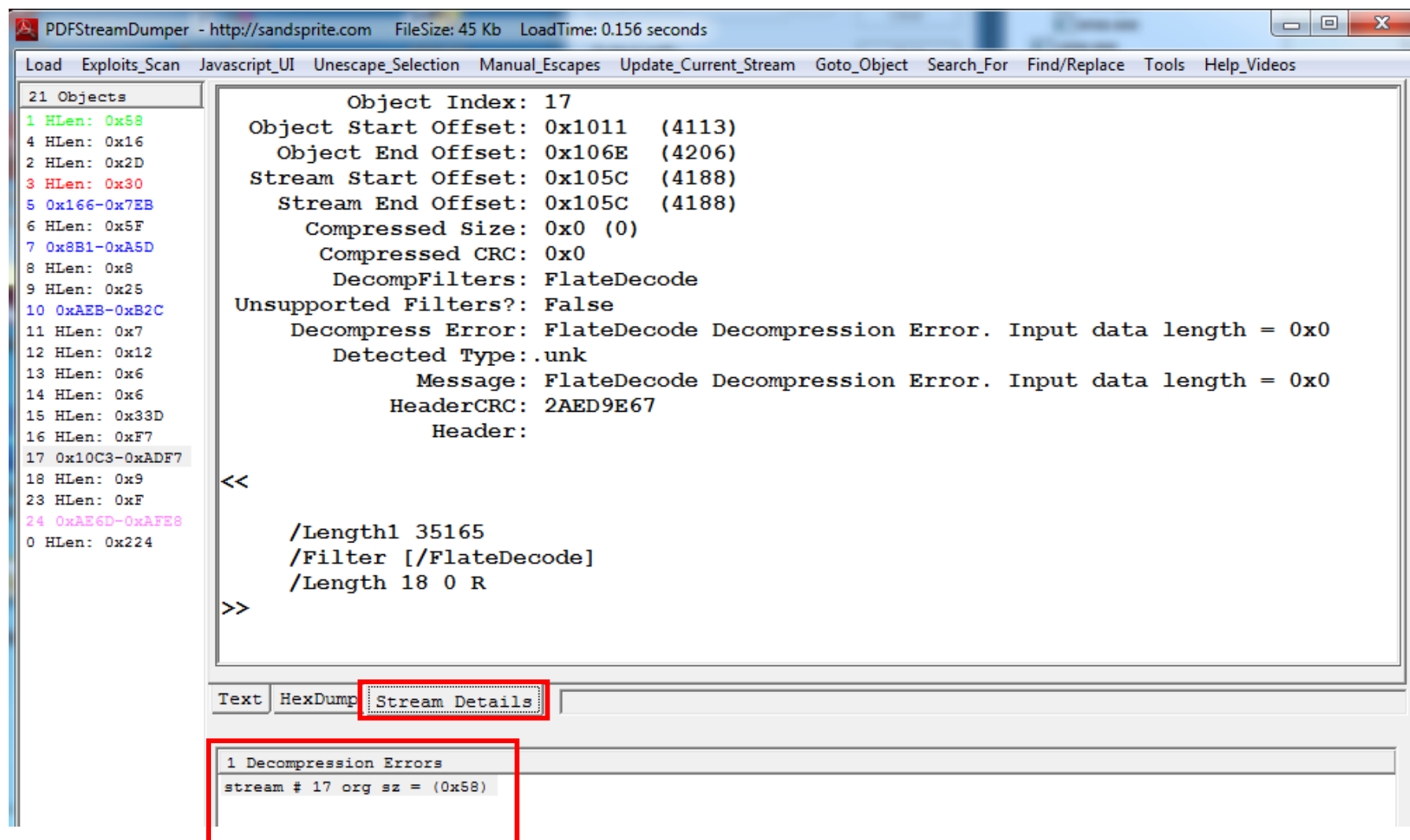
Streams:5 JS: 1 Embeds: 0 Pages: 0 TTF: 1 U3D: 0 flash: 0 UnkFlt: 1 Action: 1 PRC: 0

PDFStreamDumper

 Red: Headers with Javascript tag
Blue: Object Streams
Green: Headers with /Launch or /Action or /OpenAction or /AA
Purple: Headers with /EmbeddedFiles
Orange: Unsupported Filters
Yellow: TTF Fonts
Pink: XML Data

OK

PDF Stream Dumper



pdfid

```
remnux@remnux:~/Desktop$ pdfid sam1.pdf
PDFiD 0.2.1 sam1.pdf
PDF Header: %PDF-1.4
obj 21
endobj 21
stream 6
endstream 6
xref 1
trailer 1
startxref 1
/Page 1
/Encrypt 0
/ObjStm 0
/JS 1
/JavaScript 1
/AA 0
/OpenAction 1
/AcroForm 1
/JBIG2Decode 0
/RichMedia 0
/Launch 0
/EmbeddedFile 0
/XFA 1
/Colors > 2^24 0
```

peepdf

- Can pass it arguments to get a quick overview, or drop into an interpreter to ask more advanced questions.
- Already installed in remnux distro.

```
remnux@remnux:~/Desktop$ peepdf sam1.pdf -i
File: sam1.pdf
MD5: 27eea8d0ecf2dc7217e76a7ec98de90f
SHA1: 6598948fad8890be8981cb47404e14e7873d4746
Size: 45600 bytes
Version: 1.4
Binary: True
Linearized: False
Encrypted: False
Updates: 0
Objects: 21
Streams: 6
Comments: 0
Errors: 1
```

```
Version 0:
Catalog: 1
Info: 4
Objects (21): [1, 2, 3, 4, 5, 6, 7, 8, 9
Streams (6): [24, 5, 7, 10, 17, 17]
  Encoded (4): [5, 7, 17, 17]
Suspicious elements:
  /AcroForm: [1]
  /OpenAction: [1]
  /XFA: [23]
  /JS: [3]
  /JavaScript: [3]
```

peepdf

```
PPDF> object 3
```

```
<< /Type /Action  
/S /JavaScript  
/JS 7 0 R >>
```


Demo

PDF 2

Resources for More:

- **Malicious Document Cheat Sheet:** <https://zeltser.com/analyzing-malicious-documents/>
- **oletools:** <https://github.com/decalage2/oletools>
- **officemalscanner:** <https://medium.com/@mbromileyDFIR/malware-monday-officemalscanner-b1e5f6417df6>
- **peepdf:** <https://zeltser.com/peepdf-malicious-pdf-analysis/>
- **PDF Stream Dumper:** <https://zeltser.com/pdf-stream-dumper-malicious-file-analysis/>
- **PDF Overview:** <https://blog.didierstevens.com/2010/09/26/free-malicious-pdf-analysis-e-book/>
 - Pure Gold.

Lab Work

- Malicious Word Document Lab
- Malicious PDF Lab
 - Encompasses malicious JS
- **LAST AFTERNOON** to work on Obfuscated Malware Lab!

Sources/Questions/Comments/Corrections

- Note that animations (mostly highlighting or revealing code on click) are extremely useful when teaching from this slide deck. Email me for slide originals.
- Questions/Comments/Corrections to Lauren Pearce – Laurenp@lanl.gov